Lecture 4: Network Science review using Networkx

Let’s start with the Participation of Students

https://docs.google.com/presentation/d/1x5bsd1pEd2hGmFc3VjiTE_Z9ASTv8SCRN6x0CmsZRIA/edit?usp=sharing

Today

- We present Watts Strogatz model in NetworkX
- We present Random Graph model mentioned in the article
- Show how to read a Social Network, extract shortest paths between nodes

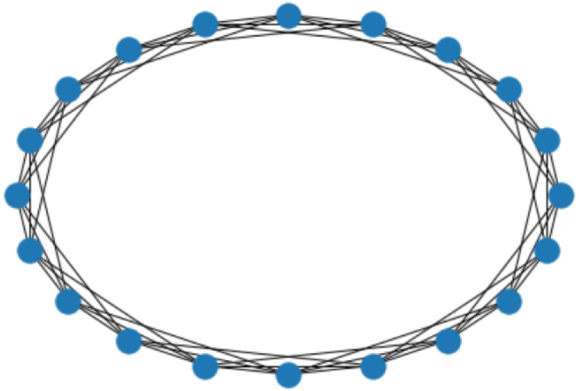
SW_Model.ipynb

Note the model always take the smallest even integer you pass:

Small-World Networks

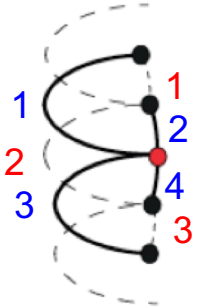
Let's reproduce Fig 1 of the article

```
: 1 G_s=nx.watts_strogatz_graph(20,7,0)

: 1 np.mean(list(dict(G_s.degree()).values()))

: 6.0

: 1 nx.draw_circular(G_s)
```



Calculate the Clustering coefficient for the red node:

Jasper Lee



$$C_i = \frac{2e_i}{k_i(k_i - 1)}$$

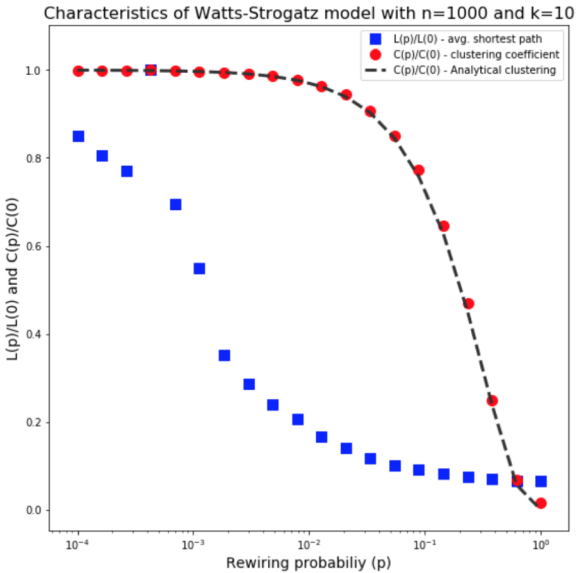
Where:
e_i = # of edges between i's neighbors
k_i = degree of node i

$e_{red} = 3$

$k_{red} = 4$

$$C_{red} = \frac{2e_{red}}{k_{red}(k_{red} - 1)} = \frac{2 \times 3}{4 \times 3} = \frac{1}{2} = \mathbf{0.5}$$

Q8: In the plot below the dashed line is `(1-p)**3.0` can you explain why that is the case? (see SW_Model.ipynb)



Daniel Grieb

The analytical formula for the clustering coefficient in the SW model is C(p)=C(0)*(1-p)^3.

The question is to explain the formula, why would it be (1-p)^3?

Example_AnalyzingSocialNetwork.ipynb

Based on the description of fl, node how edges are added:

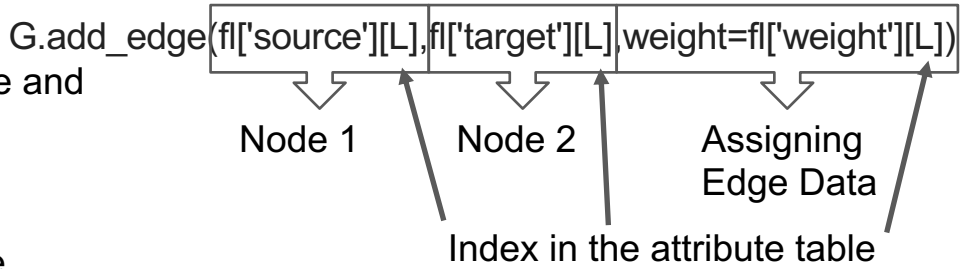
```
for L in fl.index.values:
    G.add_edge(fl['source'][L],fl['target'][L],weight=fl['weight'][L])
```

See: https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.Graph.add_edge.html

fl is an attribute table that contains a source node and a target node and their associated weight.

The code will start at the first row of this attribute table and add an edge between the given source node and the given target node and insert the weight as edge data using the keyword 'weight'. After the first row, it the code will connect the nodes of the second row and so forth until the source and target nodes in all the rows are connected.

This data is added to the DiGraph G




```
In [17]: fl
```

Out[17]:

	source	target	type	id	label	weight
0	3	37	Directed	2127	NaN	1.0
1	3	41	Directed	2128	NaN	1.0
2	3	105	Directed	2129	NaN	1.0
3	4	35	Directed	2130	NaN	2.0
4	4	62	Directed	2131	NaN	1.0
...	...	...	...	...	...	...
452	110	29	Directed	2579	NaN	5.0
453	110	60	Directed	2580	NaN	3.0
454	110	62	Directed	2581	NaN	5.0
455	110	80	Directed	2582	NaN	3.0
456	110	97	Directed	2583	NaN	6.0

457 rows × 6 columns

See how isolated nodes are eliminated:

From the example file:

```
deg = G.degree()
to_keep = []
for node in G.nodes():
        if deg[node] != 0:
        to_keep.append(node)
        else:
        print("Node: ",node," degree:
",deg[node])
#Create the network only with connected nodes
G=G.subgraph(to_keep)
```

Although, a cleaner way to do this is probably:

G.subgraph(n for n, d in G.degree if d > 0)

Check the relation between:
number of nodes, number of links, average degree and sum of the degree of all nodes

In [43]:
```
#Number of nodes and links of G
NumNodes = G.number_of_nodes()
NumEdges = G.number_of_edges()
print("Number of nodes: ",NumNodes)
print(("Number of links: ",NumEdges))
```

```
Number of nodes:  108
('Number of links: ', 378)
```

Question 5: Write the average in_degree, out_degree, and degree of the network

In [47]:
```
print("Average degree: ",np.mean(list(dict(G.degree()).values())))
```

```
Average degree:  7.0
```

In [52]:
```
(2*G.number_of_edges())/G.number_of_nodes()
```

Out[52]:
```
7.0
```

Here we extract the list connected components

see documentation here:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.connected_components.html#networkx.algo

In [70]:
```
1 S = [G.subgraph(c).copy() for c in sorted(nx.connected_components(G.to_undirected()), key=len, reverse=True)]
```

In [71]:
```
1 len(S) #number of connected components
```

Out[71]: 1

In [72]:
```
1 S[0].number_of_edges()
```

Out[72]: 378

In [73]:
```
1 S[0].number_of_nodes()
```

Out[73]: 108

In [74]:
```
1 [e for e in S[0].edges]
```

Here we extract the list connected components

see documentation here:
https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.components.connected_components.html#networkx.algo

In [70]:
```
1 S = [G.subgraph(c).copy() for c in sorted(nx.connected_components(G.to_undirected()), key=len, reverse=True)]
```

In [71]:
```
1 len(S) #number of connected components
```

Out[71]: 1

In [72]:
```
1 S[0].number_of_edges()
```

Out[72]: 378

In [73]:
```
1 S[0].number_of_nodes()
```

Out[73]: 108

In [74]:
```
1 [e for e in S[0].edges]
```

Applications of Navigability and the Structure of Social Networks

Small World Effect

“Any individual in the world can reach any other individual through a short chain of social ties (*1*, *2*). Early experimental work by Travers and Milgram (*3*) suggested that the average
length of such chains is roughly six, and recent theoretical (*4*) and empirical (*4*–*9*)
work has generalized the claim to a wide range of nonsocial networks.

In particular, individuals in real social networks have only limited, local information
about the global social network and, therefore, finding short paths represents a nontrivial search effort (*10*–*12*).”

Source: **An Experimental Study of Search in Global Social Networks, Doods et al 2003 (attached)**

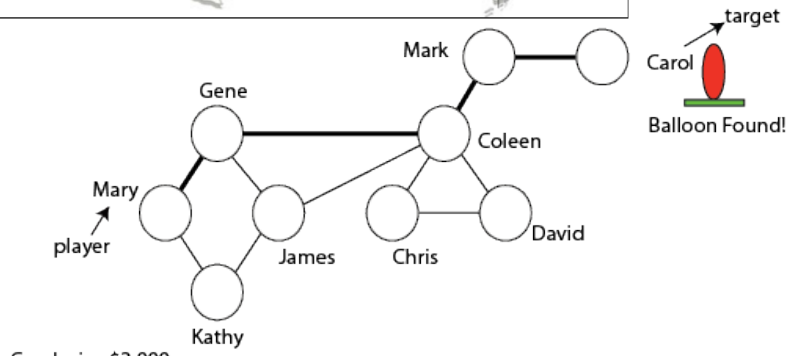
Modern version by email

Targets included a professor at an Ivy League university, an archival inspector in Estonia, a technology consultant in India, a policeman in Australia, and a veterinarian in the Norwegian
army.

Participants were informed that their task was to help relay a message to their allocated target by passing the message to a social acquaintance whom they considered "closer" than themselves to the target. Of the 98,847 individuals who registered, about 25% provided their personal information and initiated message chains.

Power of Social Network: Crowdsourcing

Red Balloon Experiment: 8 balloons randomly placed in the U.S. were found within 8 hours via social media



Carol wins $2,000
Mark wins $1,000
Coleen wins $500
Gene wins $250

```
nx.average_shortest_path_length(S[0])
```

```
3.026479750778816
```

```
G=S[0]
```

Let's find the maximun of the shortest path lengths

```
lengths = nx.shortest_path_length(G)
st_length = {}  #this will saves the maximun path per station
ml=0
for key in lengths:   #iterates in all the lengths per station
   ll = key     #only gives an intuitive name
   i = max(ll[1].values())  #finds the maximun value of length
   st_length[ll[0]] = i  #ll[0] has the name of the station and and i the value
   if i > ml:
      ml = i  #this will save the overall maximun
   print("For ", ll[0]," max length is ",i)

print("The maximun length in the network is ",ml)
```

```
For  2  max length is  4
For  3  max length is  6
For  4  max length is  5
For  5  max length is  5
For  6  max length is  5
For  7  max length is  5
For  8  max length is  5
For  9  max length is  4
For  10  max length is  5
For  11  max length is  5
For  12  max length is  5
For  13  max length is  4
For  14  max length is  4
For  15  max length is  5
```

```
The maximun length in the network is  6
```

In [97]:
```
### Add here the Paths detection
```

In [98]:
```
for key in st_length:  #iterates all lengths of each node
    if st_length[key] == ml: #if it equals to the maximun
       print(key)   #prints the name of the station
       p = nx.shortest_path(G,key)  #calculates all shorthest path from that station
       for k in p:   #iterates all the paths
          if nx.shortest_path_length(G,key,k)==ml: #finds the path that has a length
                                                   #equal to the maximum
             print(p[k])   # writes the path
```

```
3
[3, 41, 26, 30, 97, 62, 29]
27
[27, 31, 23, 4, 38, 88, 93]
29
[29, 62, 4, 35, 109, 41, 3]
[29, 62, 4, 35, 109, 41, 37]
[29, 62, 4, 23, 31, 27, 42]
[29, 62, 4, 23, 31, 47, 85]
[29, 62, 97, 30, 59, 103, 98]
37
[37, 41, 26, 30, 97, 62, 29]
42
[42, 27, 31, 23, 4, 62, 29]
[42, 94, 24, 106, 13, 88, 93]
85
[85, 47, 10, 55, 6, 110, 29]
93
[93, 88, 9, 69, 7, 50, 27]
[93, 88, 9, 69, 59, 103, 42]
[93, 88, 9, 69, 59, 103, 98]
98
[98, 103, 14, 17, 6, 110, 29]
[98, 103, 14, 68, 20, 88, 93]
```

```python
lengths = nx.shortest_path_length(G)
plengths=[]
for key in lengths:   #iterates all the kengths
    ll = key
    for i in list(ll[1].values()):    #saves a list with the lengths greater than zero
        if i > 0:plengths.append(i)
```

```python
max(plengths)
```

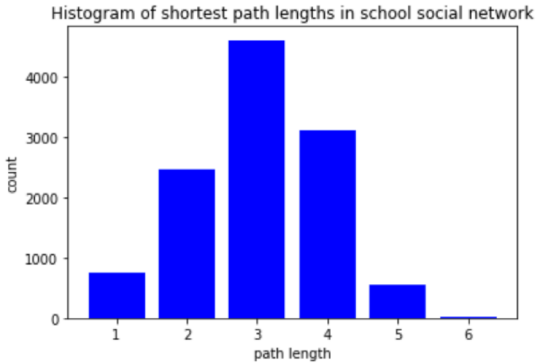
6

Histograms of Path Lengths

In [113]:
```python
counts, bins = np.histogram(plengths, bins = range(1,8))
```

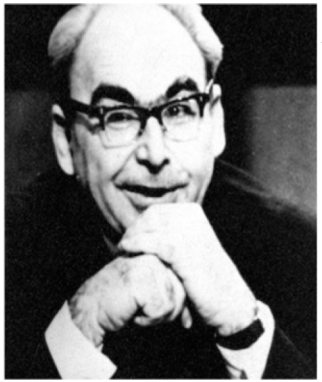
In [102]:
```python
#G1:

plt.bar(bins[:-1],counts,color='b')
plt.xlabel('path length')
plt.ylabel('count')
plt.title('Histogram of shortest path lengths in school social network')
```

Out[102]: Text(0.5, 1.0, 'Histogram of shortest path lengths in school social network')



RANDOM NETWORK MODEL

Pál Erdös
(1913-1996)



Alfréd Rényi
(1921-1970)

Erdös-Rényi model (1960)

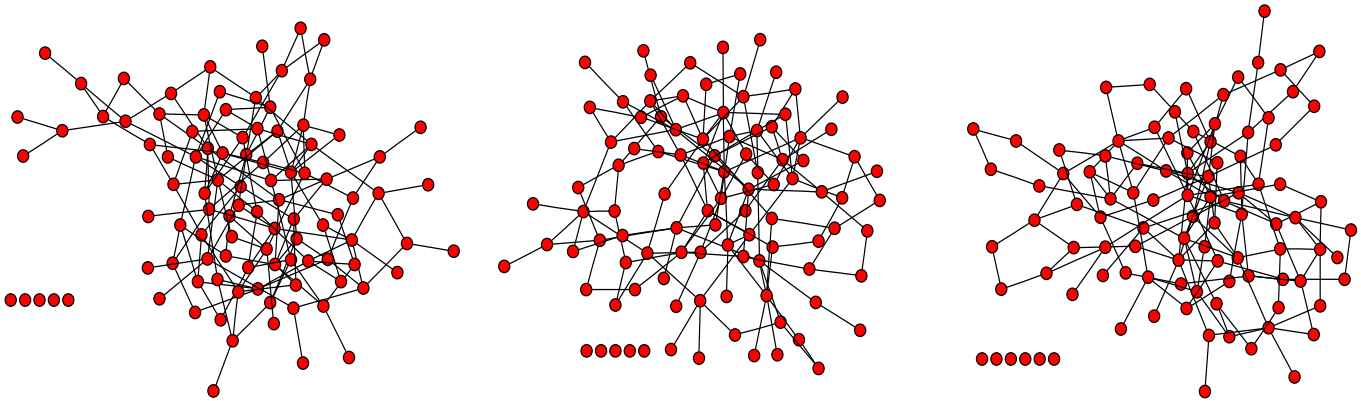
Definition:

A **random graph** is a graph of N labeled nodes where each pair of nodes is connected by a preset probability **p**.

We will call is G(N, p).

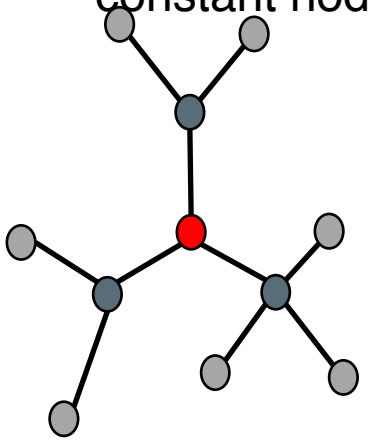
RANDOM NETWORK MODEL

p=0.03
N=100



DISTANCES IN RANDOM GRAPHS

Random graph tend to have a tree-like topology with almost constant node degrees.



- nr. of first neighbors: $\mathcal{N}_1 \cong \langle k \rangle$
- nr. of second neighbors: $\mathcal{N}_2 \cong \langle k \rangle^2$
- nr. of neighbours at distance d: $N_d \cong \langle k \rangle^d$

And the estimate maximum distance:

N can be written as:

$$N = 1 + \langle k \rangle + \langle k \rangle^2 + \ldots + \langle k \rangle^d = \frac{\langle k \rangle^{d+1} - 1}{\langle k \rangle - 1} \approx \langle k \rangle^d$$

$$\Rightarrow \quad d = \frac{\log N}{\log \langle k \rangle}$$

RANDOM GRAPHS, Mathematical results first discovered by Erdos

As quantitative data about real networks became available, we can compare their topology with the predictions of random graph theory.

Note that once we have N and <k> for a random network, from it we can derive every measurable property. Indeed, we have:

Average path length: $< l_{rand} > \approx \frac{\log N}{\log \langle k \rangle}$

Clustering Coefficient: $C_{rand} = p = \frac{\langle k \rangle}{N}$ $< k >= (N-1)p$

Degree Distribution: $P_{rand}(k) \cong C_{N-1}^{k} p^{k} (1-p)^{N-1-k}$

$$P(k) = e^{-<k>} \frac{< k >^{k}}{k!}$$

RandomGraphs+Histograms.ipynb

Degree Distribution

Intro

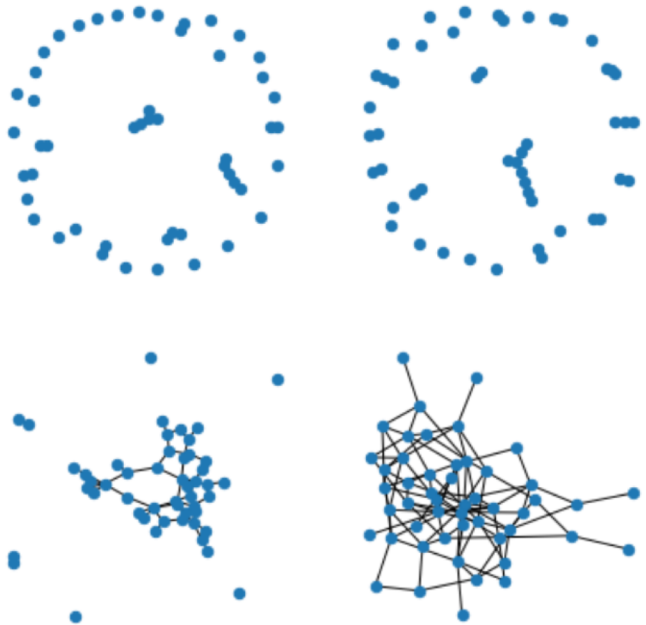
In [34]:

```
# (1.1) define random graphs
G1 = nx.erdos_renyi_graph (50,0.01) #inputs are (N,p)
G2 = nx.erdos_renyi_graph (50,0.02)
G3 = nx.erdos_renyi_graph (50,0.04)
G4 = nx.erdos_renyi_graph (50,0.08)

```

In [36]:

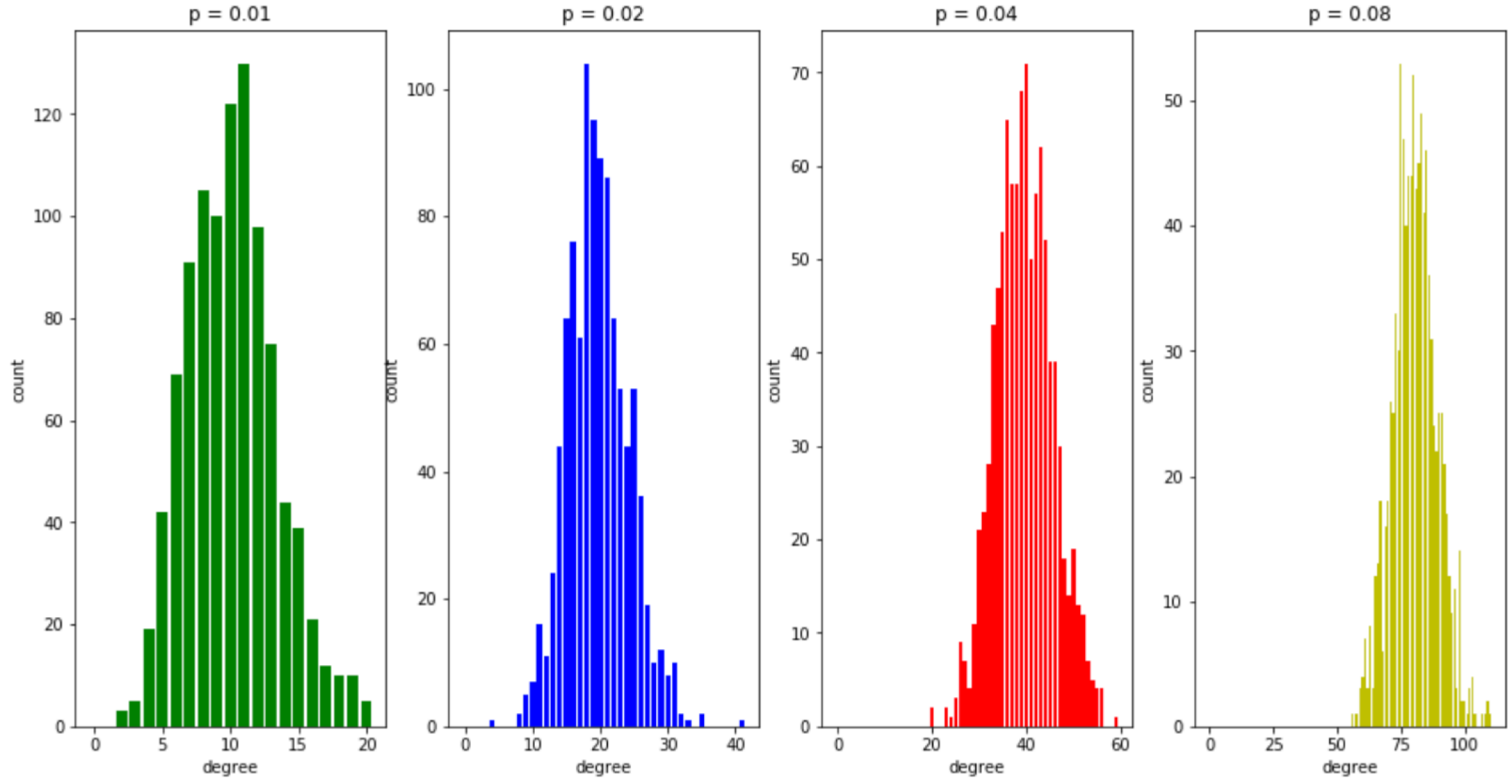
```
# (1.3) "Spring layout": use "nx.spring_layout"
fig = plt.figure(figsize=(7,7))
pos = nx.spring_layout(G4 ,iterations =1000)
plt.subplot(2,2,1)
nx.draw(G1 ,node_size =40); plt.axis('tight')
plt.subplot(2,2,2)
nx.draw(G2 ,node_size =40); plt.axis('tight')
plt.subplot(2,2,3)
nx.draw(G3 ,node_size =40); plt.axis('tight')
plt.subplot(2,2,4)
nx.draw(G4 ,node_size =40); plt.axis('tight')
```

Out[36]: (-0.602941153439874, 1.094989765840802, -0.9125935886028537, 1.099035049414855)

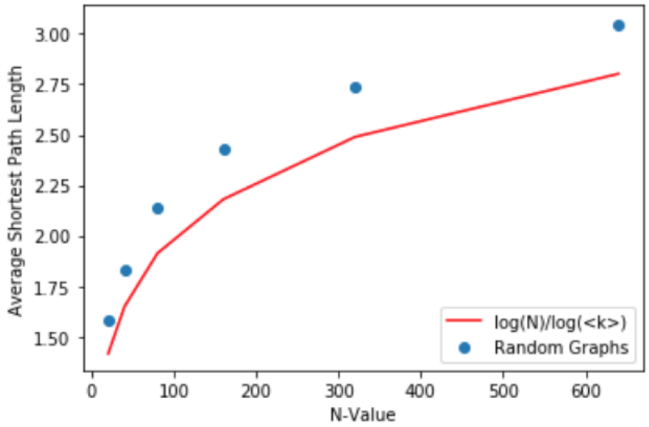


RG+degreedistributions.ipynb

```
G1 = nx.erdos_renyi_graph (1000,0.01)
G2 = nx.erdos_renyi_graph (1000,0.02)
G3 = nx.erdos_renyi_graph (1000,0.04)
G4 = nx.erdos_renyi_graph (1000,0.08)
```



Q10: Explain this plot:, fromRG+degreedistributions.ipynb



It can be demonstrated that L~ln(N)/ln(k)

With a fixed amount of total edges (10 in this .ipynb example), the **average shortest path length (ASPL)** follows the trend of L_{random}~ ln(N)/ln(<k>) in which:

N is the total number of nodes
<k> is the mean degree value of the entire graph

It is shown that: as N increases from 20 to 640, ASPL increases, while the increase rate of ASPL decreases. And ASPL is slightly over log(N)/log(<k>) for each N value.

Testing the resut of <L> ~ log(N)/log<k>

```
pathLengthVals=[]
meandegVals=[]
#numNodes=[20,40,80,160,320,640,1280,2000,4000,8000] #this may take time to run about 30min
numNodes=[20,40,80,160,320,640]
ApproxL=[]
for i in range (len(numNodes)):
    G=nx.fast_gnp_random_graph(numNodes[i],10.0/numNodes[i],seed=3,directed=False)
    pathLengthVals.append(nx.average_shortest_path_length(G))
    meandegVals.append(np.mean(list(dict(nx.degree(G)).values())))
    ApproxL.append(math.log(numNodes[i])/math.log(meandegVals[i]))
    print(numNodes[i]," ",nx.average_shortest_path_length(G)," ",ApproxL[i])
```

```
20   1.5789473684210527   1.4155815555351283
40   1.8346153846153845   1.650227472957508
80   2.141772151898734    1.9135434754184442
160   2.4265723270440254   2.1807340838850893
320   2.7356778996865203   2.4897173726420245
640   3.0460974178403757   2.8023825940022467
```

```
plt.scatter(numNodes,pathLengthVals,label='Random Graphs')
plt.plot(numNodes,ApproxL,'r-',label='log(N)/log(<k>)')
plt.xlabel('N-Value')
plt.ylabel('Average Shortest Path Length')
plt.legend(loc='lower right')
```

```
<matplotlib.legend.Legend at 0xa15761f90>
```


```
plt.scatter(numNodes,pathLengthVals,label='Random Graphs')
plt.plot(numNodes,ApproxL,'r-',label='log(N)/log(<k>)')
plt.xlabel('N-Value')
plt.ylabel('Average Shortest Path Length')
plt.legend(loc='lower right')
```


```
<matplotlib.legend.Legend at 0xa15761f90>
```